Diplomvortrag: Verteiltes Debugging

Gemeinsames Debuggen in Saros

Umut Erdogan
Eingereicht bei Prof. Lutz Prechelt
Betreuer: Dr. Karl Beecher

Was ist ein Debugger?

- Programm (Debugger), welches die Ausführung eines anderen Programmes (Debuggee) steuert.
- Erlaubt Analyse des dynamischen Verhaltens
- Ausführung kann
  - in Einzelschritten erfolgen (Step-Anweisung)
  - an gesetzten Haltepunkten (Breakpoints) stoppen
  - In Java: bei Zugriff auf beobachtete Feldelemente (Watchpoints) stoppen
- Werte von Variablen nur im Stoppzustand abgefragt und gesetzt werden

Was ist ein verteilter Debugger?

- Debugger erweitert um die Möglichkeit parallel von mehreren Benutzern gleichzeitig bedient zu werden
- Impliziert Verteiltheit der Anwendung wegen mehrfacher Ein- und Ausgabegeräte
- Bedarf Berücksichtigung der Anforderungen an Groupware (Datenhaltung, Parallelität, Synchronisation,Konfliktmanagement,Awareness)
- Benutzer können Scheduling von Threads übernehmen: Effizientes testen von Ablaufszenarien.

Motivation

- Saros ist Groupware zur Manipulation von Quelltext. Statische Bearbeitung mit Erkennung von Fehlern auf Compiletime-Ebene.
- Anwendungslogik von Programmen sollen ebenfalls mit Saros untersucht werden können.
- Nebenläufigkeit von Programmen (Multithreading) ist interne Quelle für Programmdynamik. Externe Quelle ist z.B. Kommunikation mit externem Prozeß.
- Nebenläufigkeit von nichtdeterministischer Natur (Scheduler). Wunsch kritische Situationen in Szenarien deterministisch zu erzeugen und zu analysieren.
- Debugger als Werkzeug zur Analyse des Dynamischen Verhaltens. Problem: Nur für Single-User Betrieb konzipiert.

Ziel:Multi-User-Debugger

- Debugger als Realisierung einer Multi-User Anwendung in einer verteilten Benutzerumgebung integriert in Saros.
- Individuelles Abfragen von Debugging-Strukturen soll effizienteren Einblick in Programmzustand ermöglichen
- Breakpoints und Watchpoints erstellt und unter den Benutzern ausgetauscht werden können.
- Beobachtung und Steuerung von Threads durch verschiedene Benutzer (abwechselnd oder simultan) erlaubt RunTime-Brainstorming
- Parallele Abläufe durch Threadzuweisung an verschiedene Benutzer analysierbar auf Sonderfälle, Engpässe, Deadlocks

Herangehensweise

- Debugging Framework von Eclipse verstehen und am Beispiel des Java Debuggers Realisierbarkeit von Multi-User Betrieb untersuchen.
- Saros mit Multi-User Java Debugger ausstatten, welches möglichst alle Merkmale des Ausgangsdebuggers besitzt und um Aspekte wie Awareness, Multi-User User-Interfaces, Unabhängigkeit von Viewports u.a. erweitert ist
- Anwenden der Erkenntnisse zum Umbau für Debugger anderer Programmiersprachen

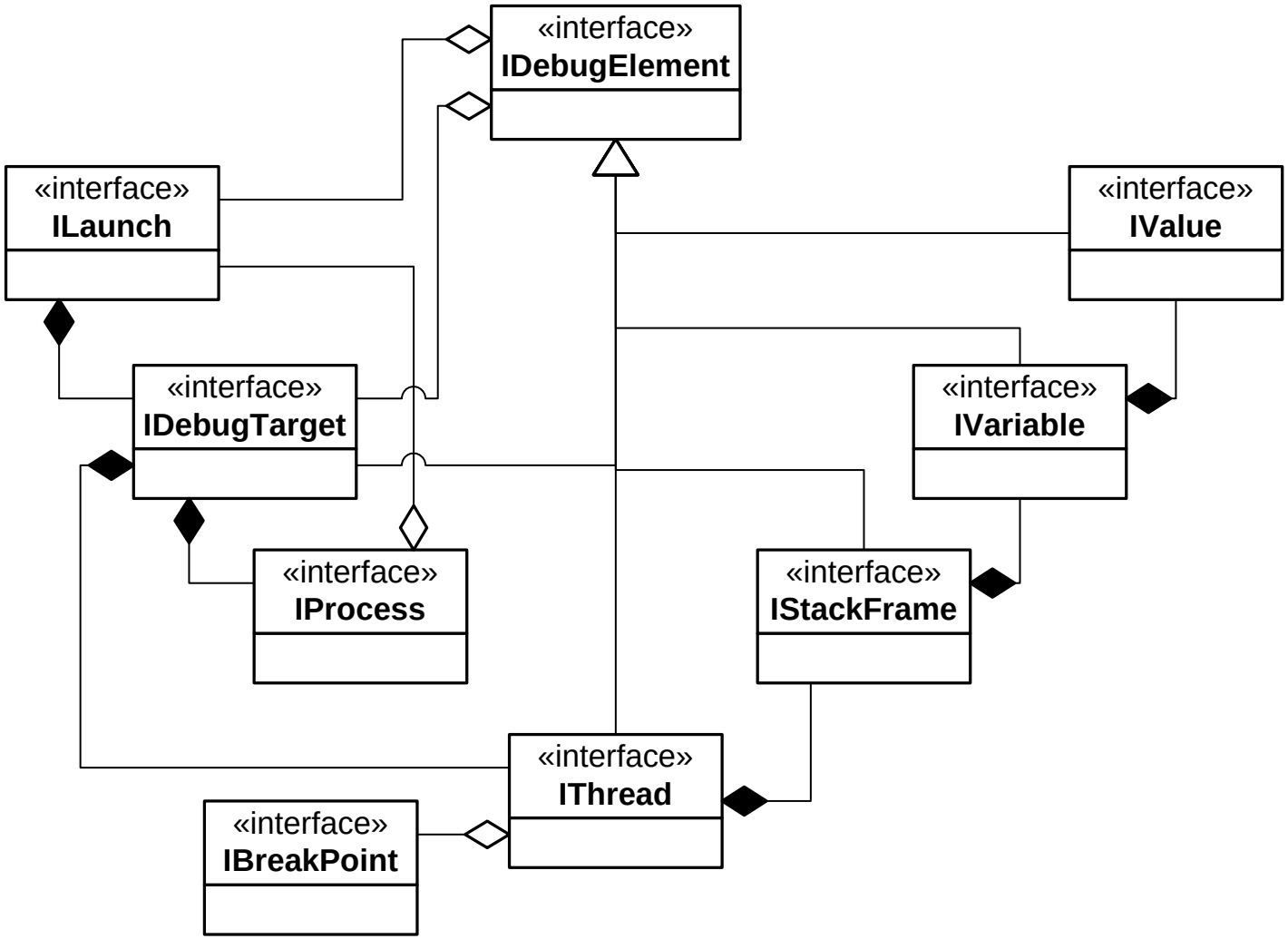
Methoden

- Generische Eclipse Standard Debug Platform verstehen
- Spezifikation der Java Platform Debugger Architecture (JPDA) verstehen
- Java Debugger in Runtime Eclipse einsetzen und dabei Quellcode des Java Debuggers schrittweise bei der Arbeit beobachten
- Anforderungen und Methoden aus anderen Multi-User Applikationen auswählen und Lastenheft aufstellen
- Saros Infrastruktur und Kommunikationsprotokoll XMPP anpassen.
- Probleme der Serialisierung und Synchronisation lösen -> Mailinglisten von Eclipse und Xstream (Serialisierungs-Framework)

Eclipse Standard Debug Model(ESDM)

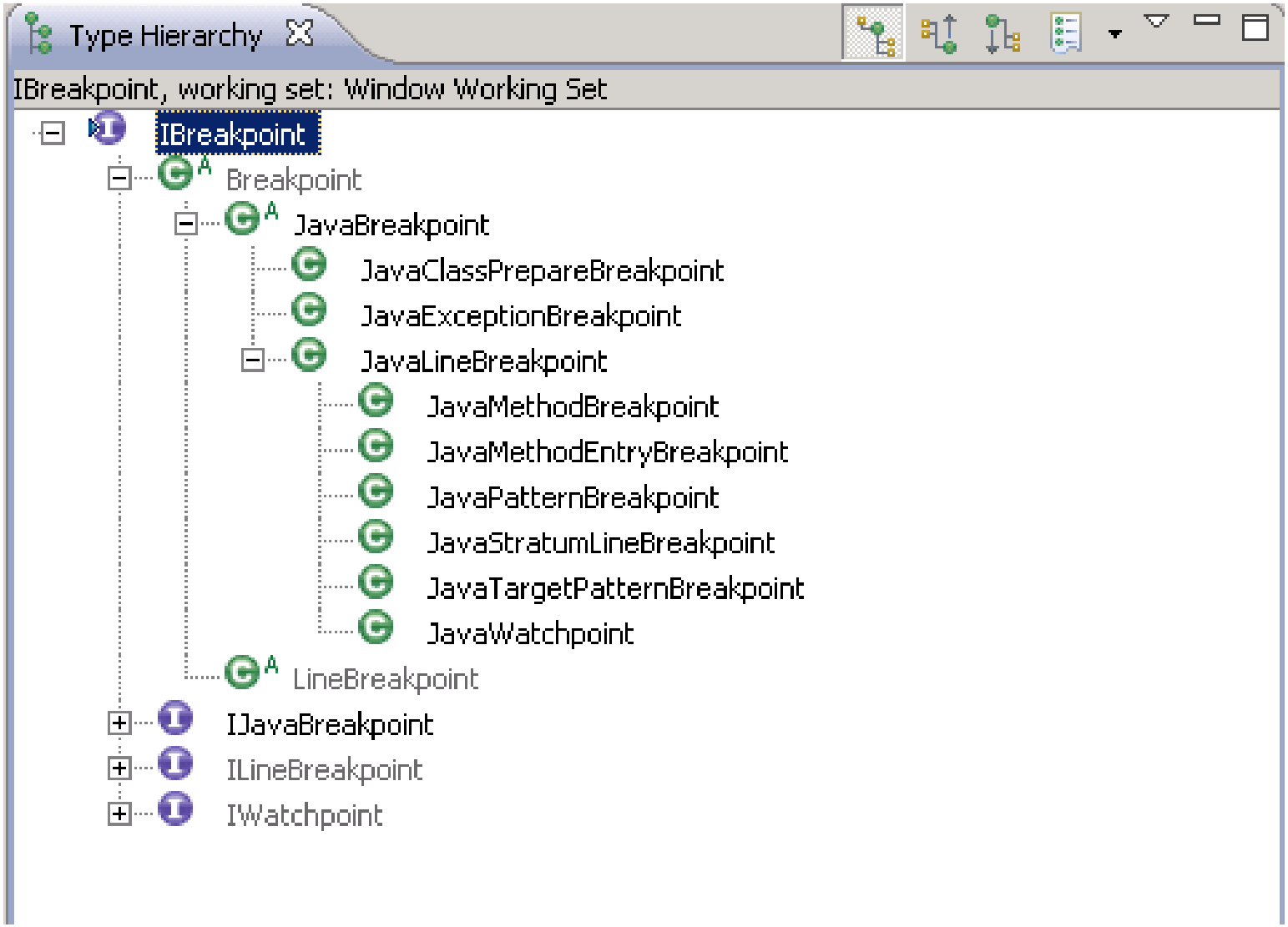
- Abstraktes Datenmodell eines Debuggees einer imperativen Sprache.
- Kern der Eclipse Debug Platform
- Eclipse-Debug-Perspektive kommuniziert mit diesem abstrakten Datenmodell ->Generische GUI
- Breakpoints als Aggregat von Threads, wenn diese Thread-Suspend ausgelöst haben
- Konkreter Debugger implementiert Interfaces des Datenmodells und erhält dadurch GUI

Eclipse Standard Debug Model (ESDM)

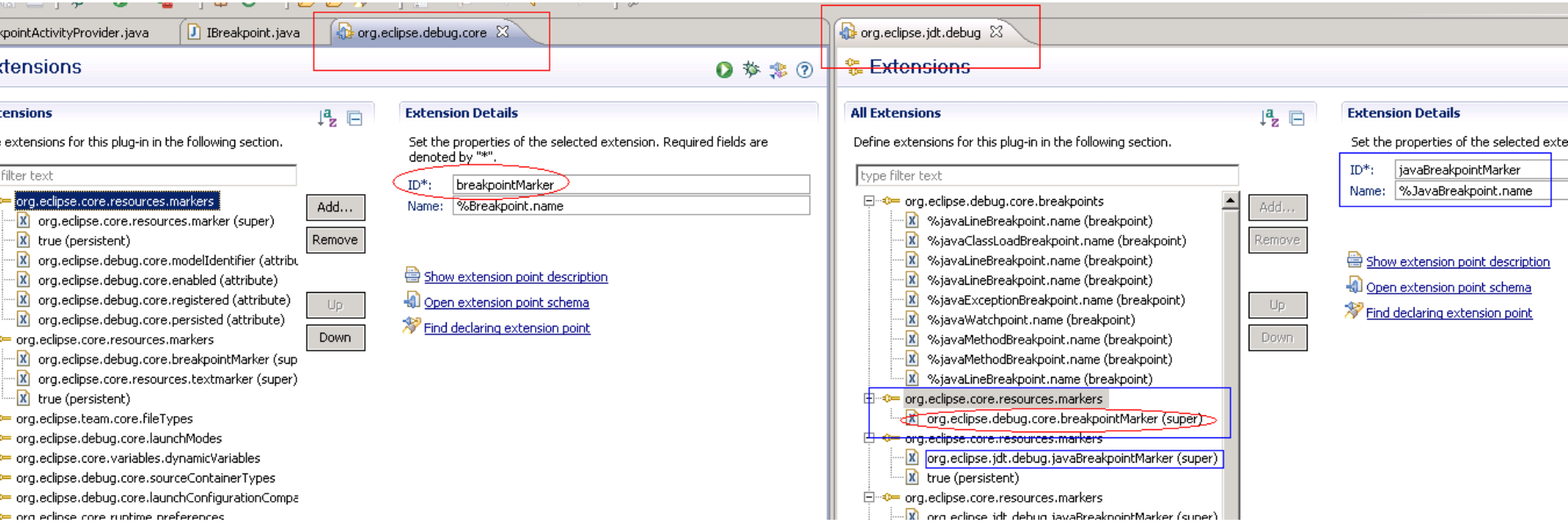


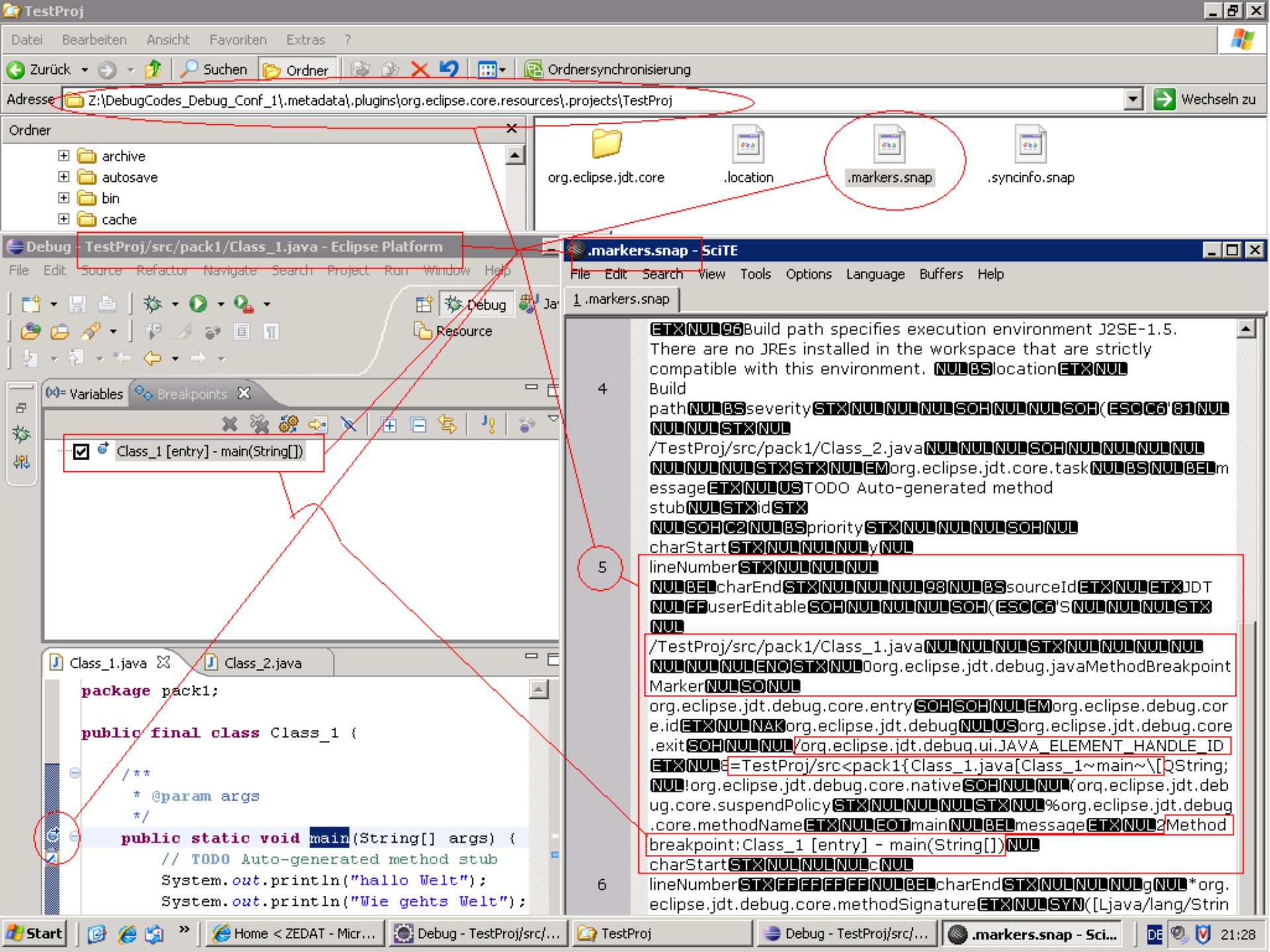
http://www.eclipse.org/eclipse/debug/documents/ec_2008/Debug_Tutorial_Basic_2008.ppt

Eclipse: Breakpoints in Java



Eclipse:
1:1 assoziierte Marker zu Breakpoints




TestProj
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Ordner Ordnersynchronisierung
Adresse Z:\DebugCodes_Debug_Conf_1\.metadata\.plugins\org.eclipse.core.resources\.projects\TestProj
Wechseln zu
Ordner
archive
autosave
bin
cache
org.eclipse.jdt.core
.location
.markers.snap
.syncinfo.snap
Debug - TestProj/src/pack1/Class_1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Resource
Variables
Breakpoints
Class_1 [entry] - main(String[])
Class_1.java
Class_2.java
package pack1;
public final class Class_1 {
/**
* @param args
*/
public static void main(String[] args) {
// TODO Auto-generated method stub
System.out.println("hallo Welt");
System.out.println("Wie gehts Welt");
.markers.snap - SciTE
File Edit Search View Tools Options Language Buffers Help
1 .markers.snap
4
5
6
Build path specifies execution environment J2SE-1.5. There are no JREs installed in the workspace that are strictly compatible with this environment.
/TestProj/src/pack1/Class_2.java
org.eclipse.jdt.core.task
message
TODO Auto-generated method stub
priority
charStart
lineNumber
charEnd
sourceId
JDT
userEditable
/TestProj/src/pack1/Class_1.java
org.eclipse.jdt.debug.javaMethodBreakpointMarker
org.eclipse.jdt.debug.core.entry
org.eclipse.debug.core.id
org.eclipse.jdt.debug
org.eclipse.jdt.debug.core.exit
/org.eclipse.jdt.debug.ui.JAVA_ELEMENT_HANDLE_ID
=TestProj/src<pack1{Class_1.java[Class_1~main~\[QString;
!org.eclipse.jdt.debug.core.native
(org.eclipse.jdt.debug.core.suspendPolicy
%org.eclipse.jdt.debug.core.methodName
main
message
Method breakpoint:Class_1 [entry] - main(String[])
charStart
lineNumber
charEnd
org.eclipse.jdt.debug.core.methodSignature
([Ljava/lang/Strin
Start
Home < ZEDAT - Micr...
Debug - TestProj/src/...
TestProj
Debug - TestProj/src/...
.markers.snap - Sci...
21:28

Umsetzung Multi-User Debugger

- Unterscheidung in Pre-Launch- und Launch-Phase
- Pre-Launch: Austausch von Breakpoints und deren Manipulation verwaltet in lokalen Breakpointmanagern
- Launch: Host allein startet Debugger und Peers folgen nach dem Prinzip von Remote Java Application launch
- Launch-Phase unterstützt alle Funktionen wie Step,Pause,Resume,Variablenabfrage usw. . Fkts.-umfang ist echte Obermenge aus der Pre-Launch Phase.

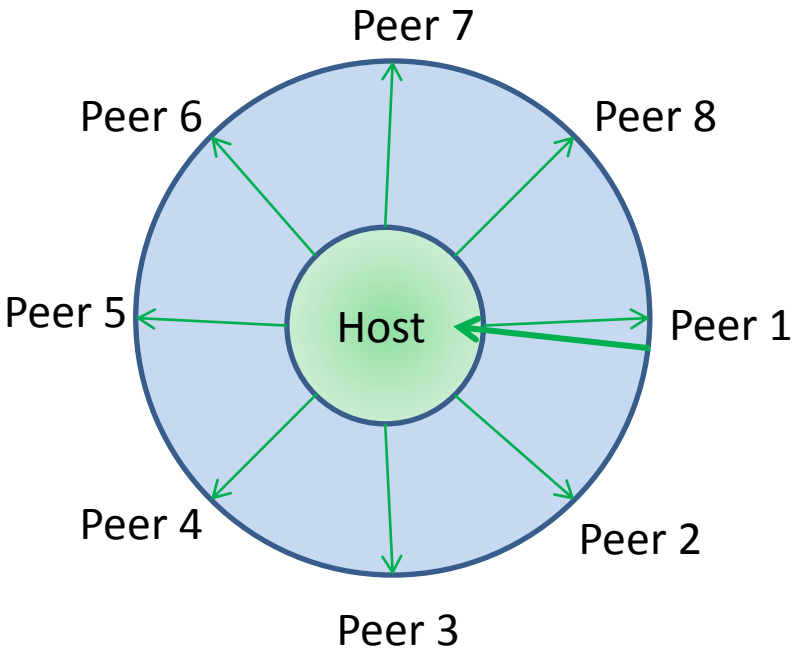
Assoziation von Breakpoints

- Peers und Host müssen sich auf gemeinsame Breakpointobjekte beziehen können, damit zu replizierende Events verarbeitet werden können
- Breakpoints sind Marker zugeordnet, welche deren Attribute persistent verwalten
- MarkerID identifiziert Marker dateilokal
- BreakpointID=(MarkerID,FilePath) systemlokal
- Peers besitzen bijektive Abb. BID_{peer}-> BID_{host}

Serialisierung von Breakpoints

- Generisch für Basistyp IBreakpoint implementiert. Daher für beliebige Programmiersprachen einsetzbar.
- Emuliert Eclipse internen Mechanismus zur persistenten Speicherung v. Breakpoints über Marker.
- Xstream als Framework kombiniert mit Javas Serialisierungsschema, deren Hook Methoden benutzt werden um in Deserialisierung geeignete Methoden zur Rückgewinnung des Breakpoints aus dem serialisierten Markerobjekt zu definieren.

Sternkommunikation



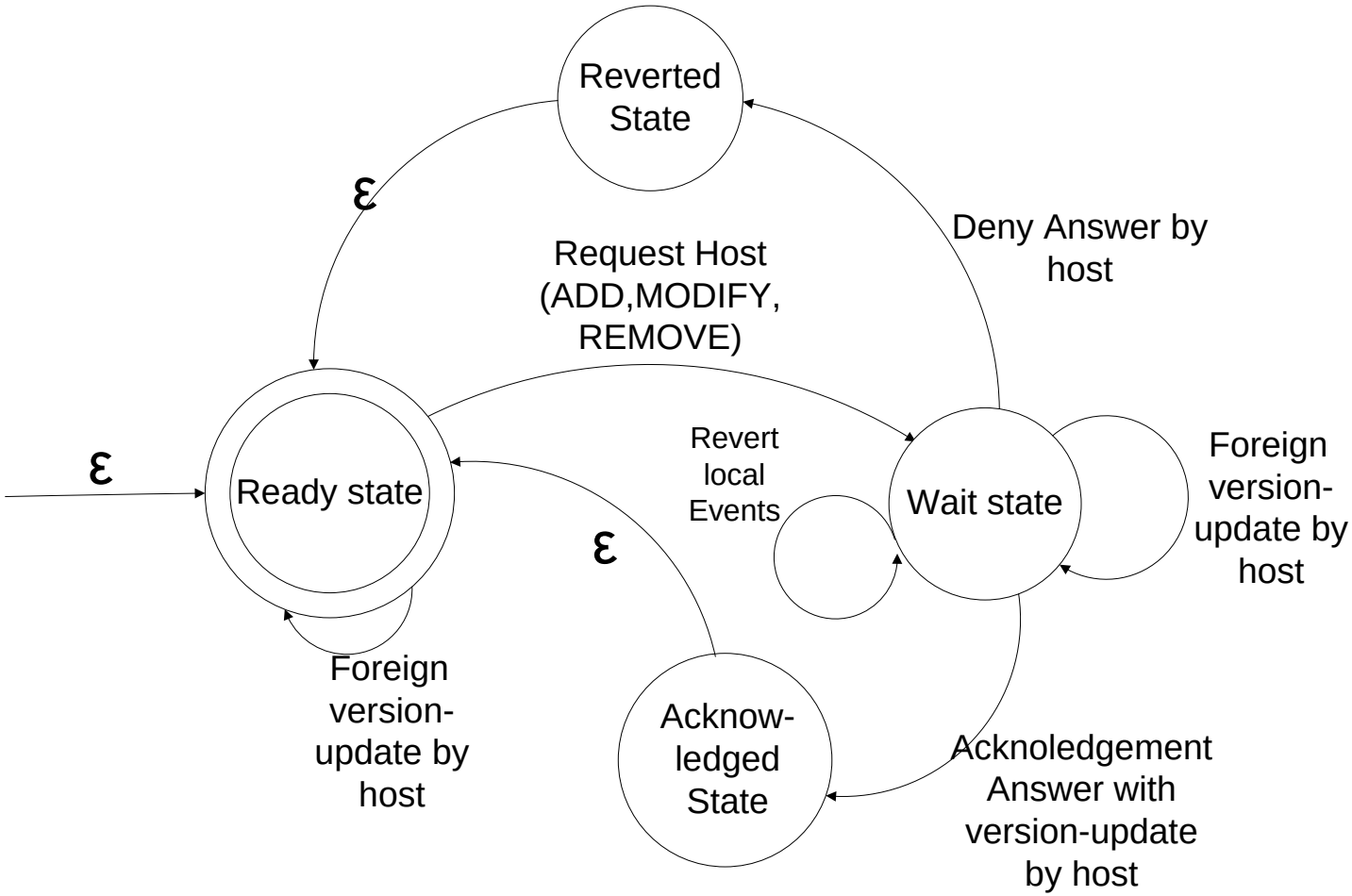
Synchronisation der Breakpointmanager

- Jeder Teilnehmer besitzt B.-Manager, welcher über UI erzeugte/manipulierte Breakpoints verwaltet.
- B.-Manager erzeugt und sendet Events, wenn sich sein Datenbestand ändert.
- SharingBreakpointListener sendet diese an Host zwecks Synchronisation
- Teilnehmer dürfen je Datei eine Änderungsaktion durchführen und müssen dann auf Rückmeldung durch den Host warten
- Es findet eine Versionszählung auf jeder Datei statt, welche vom Host zur Konflikterkennung und –auflösung ausgewertet wird. Prinzip von VCS‘en.

Konkurrenz von lokalen und empfangen Ereignissen

- Nachrichtenverarbeitung und Ausführung von BreakpointListener Methoden konkurrieren um SWT Thread
- Dadurch kann ein Peer im Wartezustand eingehende Ereignisse vom Host verarbeiten
- Host sendet verrabeitete Ereignisse an Peers und diese werden in selbiger Reihenfolge vom Peer verarbeitet
- Host muß bei Verarbeitung eines Peer Requests zwischenzeitliche Zustandsänderungen antizipieren um geeignete Konfliktlösung zu treffen

Finite State Machine of Modification Process by Peer



Behandlung von Nebenläufigkeit

- Host prüft bei Empfang von Breakpointereignis, ob Peers Dateilevel zum Zeitpunkt der Ereigniserzeugung identisch mit aktuellem Level war.
- Falls Version veraltet, prüft dieser zusätzlich, ob Konfliktpotenzial aufgrund von gleichem Breakpointtyp und Zeile mit erstellter Historie von ausgeführten Ereignissen besteht.
- Bei Durchführung der Aktion wird Breakpoint des Ereignisses in Historie gespeichert und Peer bekommt Bestätigung.
- Ansonsten Ablehnungsmitteilung an Peer: Rückgängigmachen der Aktion
- Anwendung des Prinzip der optimistischen Synchronisation aus dem Umfeld von DBMS‘en

Konfliktsituationen im zentralen BreakpointManager

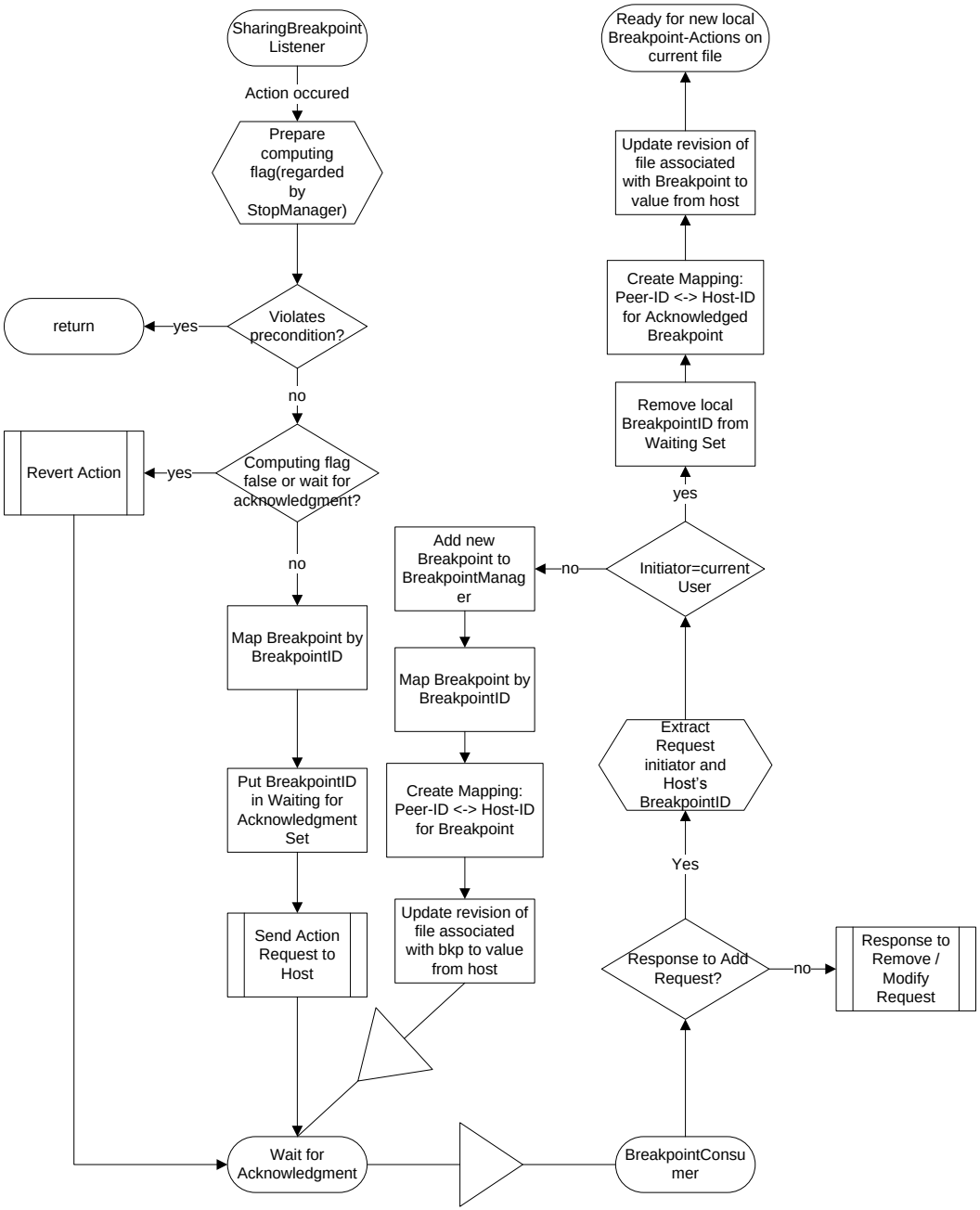
Konfliktsituationen bei parallel erzeugten Breakpointaktivitäten

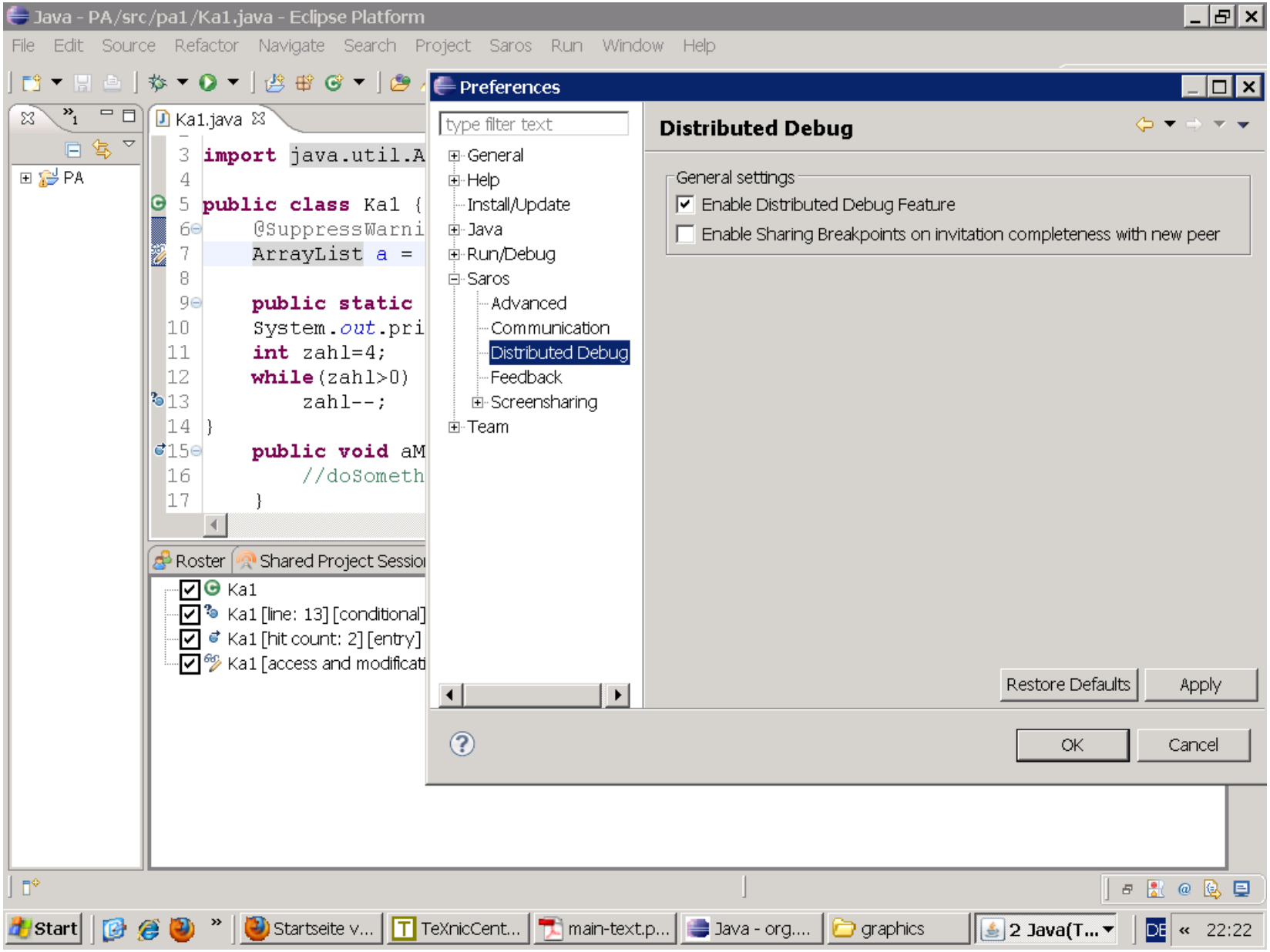
Vorereignis	Ereignis		
Gemeinsame Konfliktvorbedingung: Gleicher Typ und Gleiche Zeile bzgl. des Breakpoints der Ereignisse			
	ADD	REMOVE	MODIFY
ADD	Ereignis ablehnen.	Kann nicht vorkommen bei Konsistenz der Teilnehmer	Kann nicht vorkommen bei Konsistenz der Teilnehmer
REMOVE	Kann nicht vorkommen bei Konsistenz der Teilnehmer	Ereignis ignorieren.	Ereignis ignorieren.
MODIFY	Kann nicht vorkommen bei Konsistenz der Teilnehmer	Ereignis ablehnen.	Ereignis ablehnen.

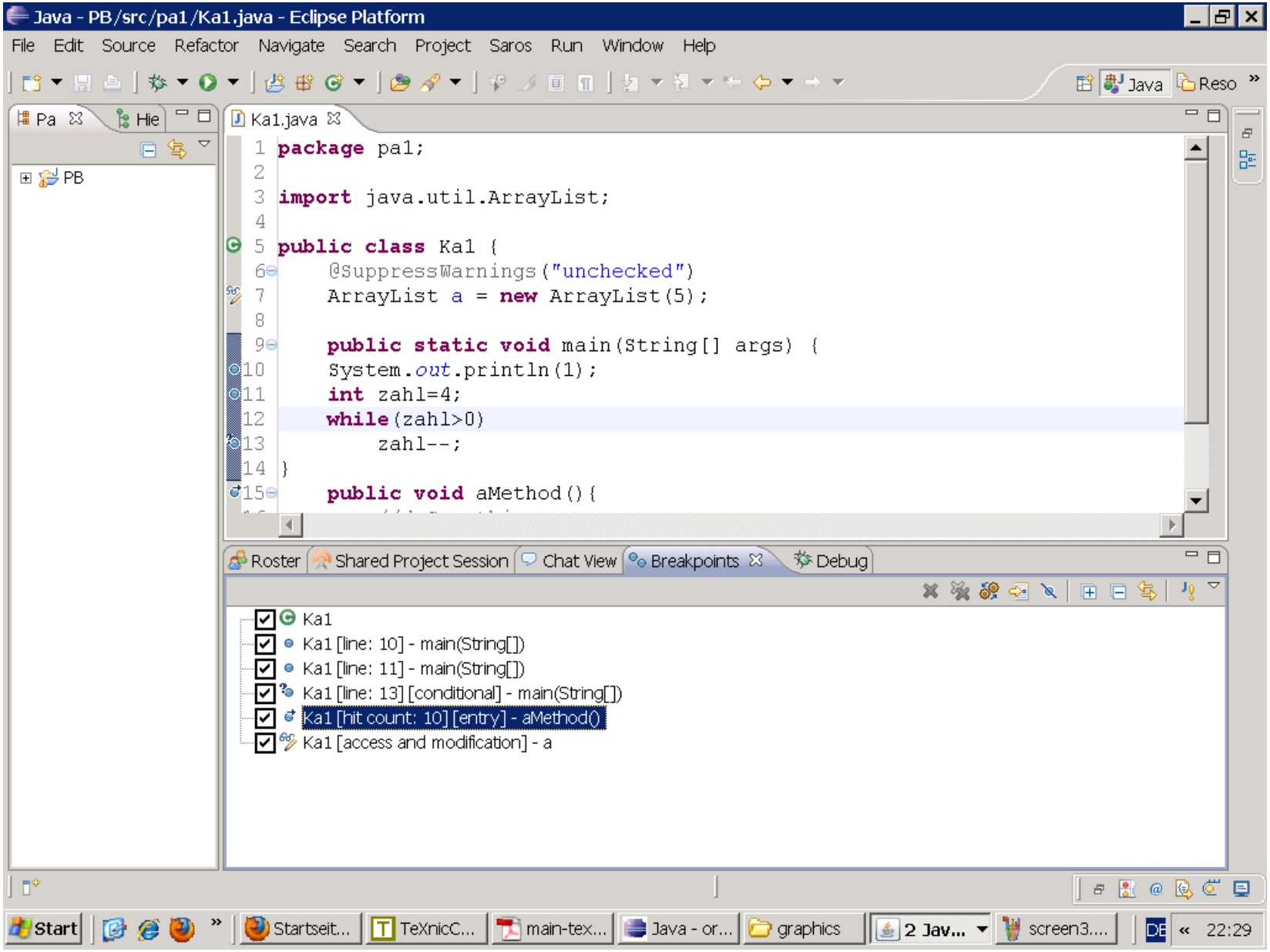
Konfliktsituationen bei MODIFY = ToggleEnable

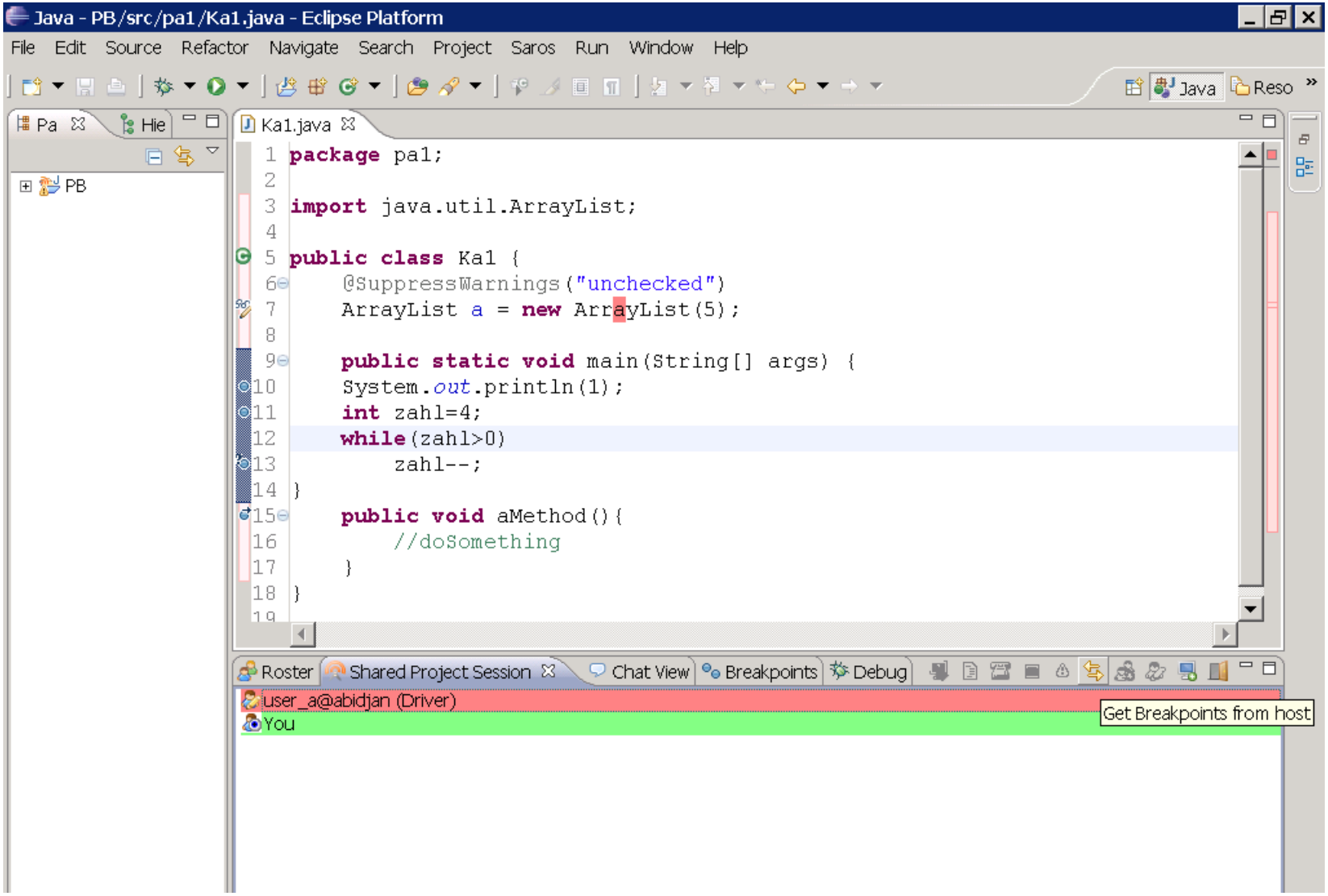
Vorereignis	Ereignis		
Gemeinsame Konfliktvorbedingung: Gleicher Typ und Gleiche Zeile bzgl. des Breakpoints der Ereignisse			
	ADD	REMOVE	ToggleEnable
ADD	Ereignis ablehnen.	Kann nicht vorkommen bei Konsistenz der Teilnehmer	Kann nicht vorkommen bei Konsistenz der Teilnehmer
REMOVE	Kann nicht vorkommen bei Konsistenz der Teilnehmer	Ereignis ignorieren.	Ereignis ignorieren.
ToggleEnable	Kann nicht vorkommen bei Konsistenz der Teilnehmer	Ereignis ablehnen.	Ereignis ignorieren.

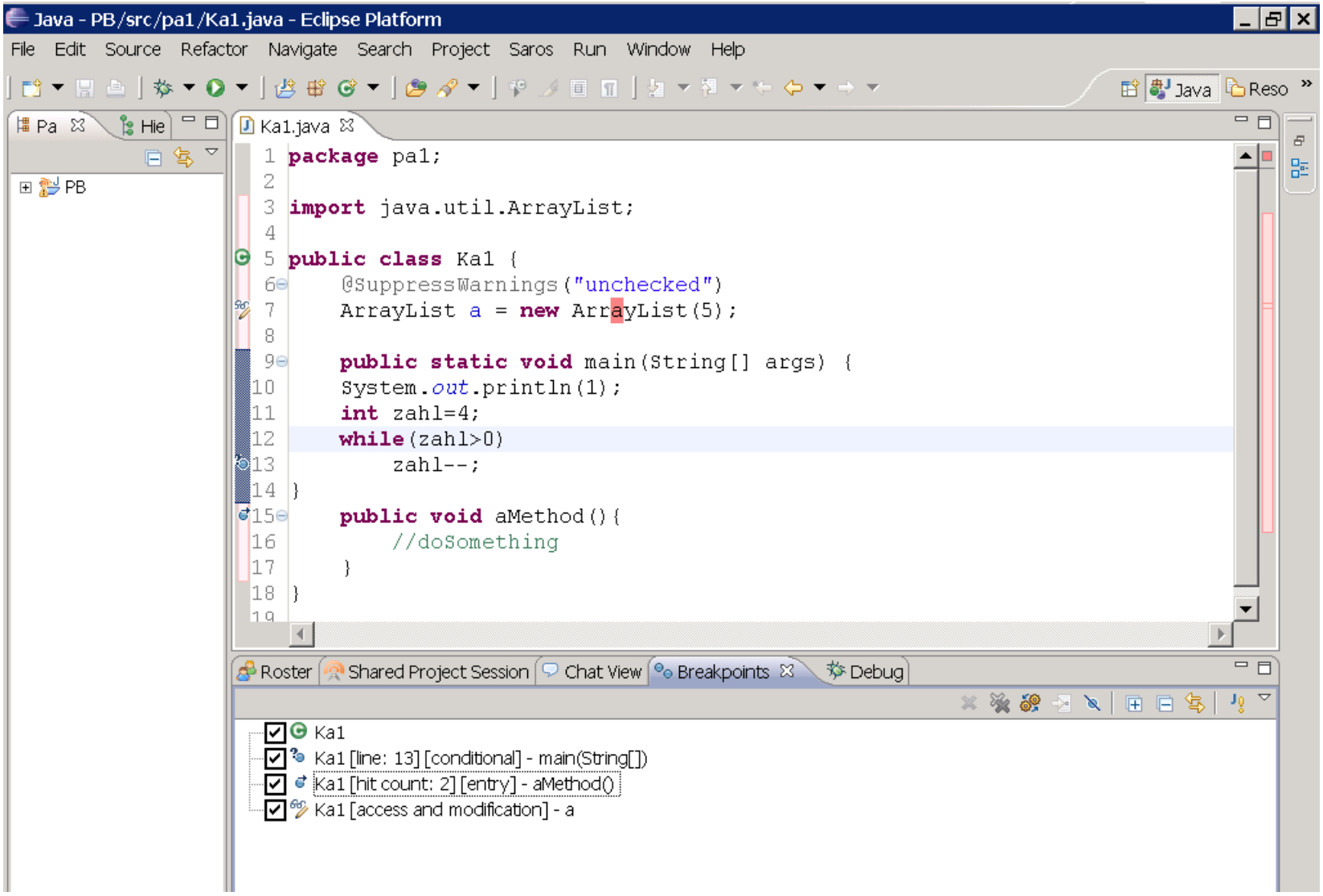
Control Flow in Scenario where Request will be Acknowledged




Java - PA/src/pa1/Ka1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Saros Run Window Help
Ka1.java
3 import java.util.A
4
5 public class Ka1 {
6 @SuppressWarni
7 ArrayList a =
8
9 public static
10 System.out.pri
11 int zahl=4;
12 while(zahl>0)
13 zahl--;
14 }
15 public void aM
16 //doSometh
17 }
PA
Roster
Shared Project Sessio
Ka1
Ka1 [line: 13] [conditional]
Ka1 [hit count: 2] [entry]
Ka1 [access and modificati
Preferences
type filter text
General
Help
Install/Update
Java
Run/Debug
Saros
Advanced
Communication
Distributed Debug
Feedback
Screensharing
Team
Distributed Debug
General settings
Enable Distributed Debug Feature
Enable Sharing Breakpoints on invitation completeness with new peer
Restore Defaults
Apply
OK
Cancel
Start
Startseite v...
TeXnicCent...
main-text.p...
Java - org....
graphics
2 Java(T...
22:22

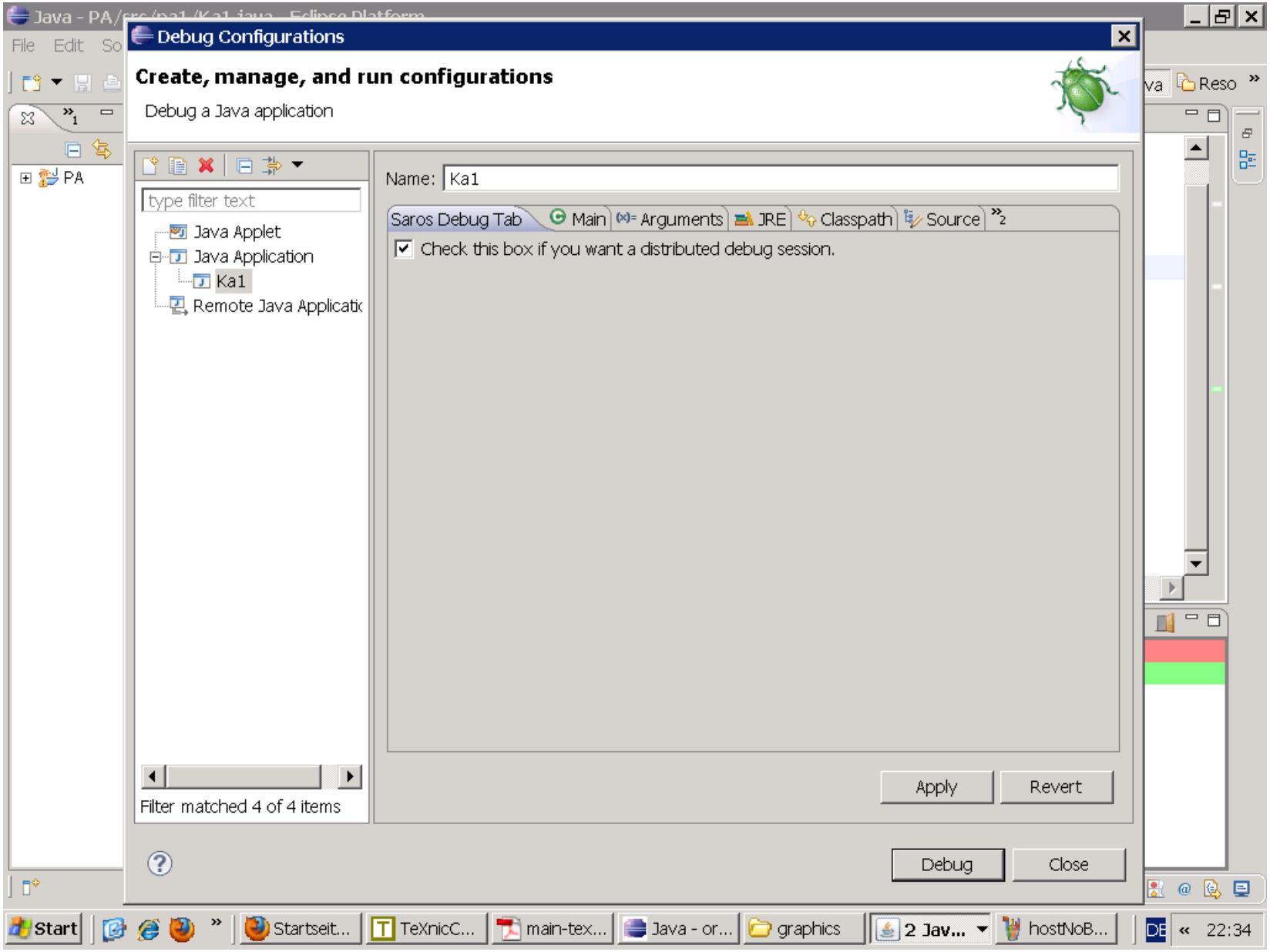

Java - PB/src/pa1/Ka1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Saros Run Window Help
Java
Reso
Pa
Hie
PB
Ka1.java
1 package pa1;
2
3 import java.util.ArrayList;
4
5 public class Ka1 {
6 @SuppressWarnings("unchecked")
7 ArrayList a = new ArrayList(5);
8
9 public static void main(String[] args) {
10 System.out.println(1);
11 int zahl=4;
12 while(zahl>0)
13 zahl--;
14 }
15 public void aMethod(){
Roster
Shared Project Session
Chat View
Breakpoints
Debug
Ka1
Ka1 [line: 10] - main(String[])
Ka1 [line: 11] - main(String[])
Ka1 [line: 13] [conditional] - main(String[])
Ka1 [hit count: 10] [entry] - aMethod()
Ka1 [access and modification] - a
Start
Startseit...
TeXnicC...
main-tex...
Java - or...
graphics
2 Jav...
screen3....
22:29

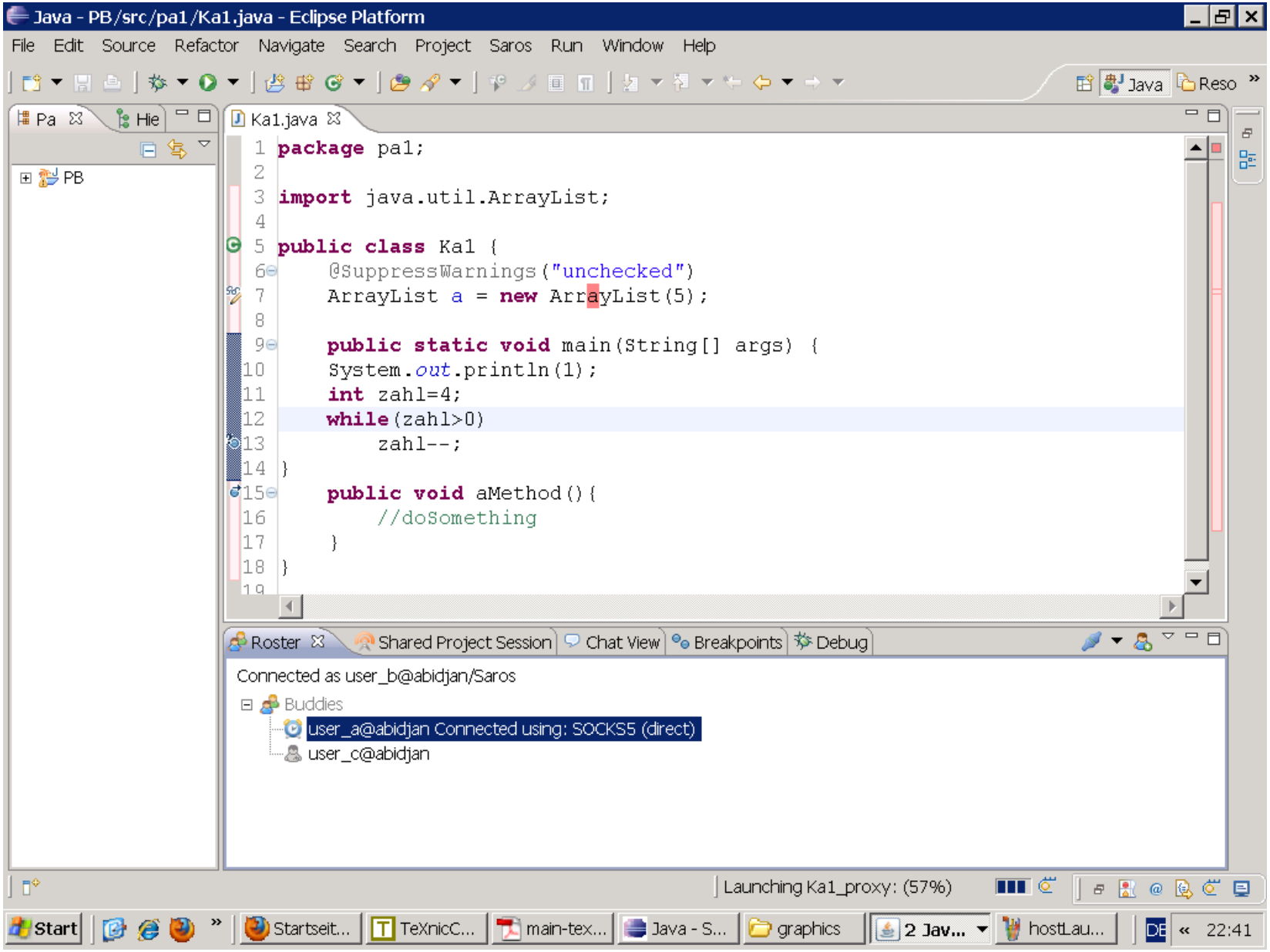

Java - PB/src/pa1/Ka1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Saros Run Window Help
Java
Reso
Pa
Hie
PB
Ka1.java
1 package pa1;
2
3 import java.util.ArrayList;
4
5 public class Ka1 {
6 @SuppressWarnings("unchecked")
7 ArrayList a = new ArrayList(5);
8
9 public static void main(String[] args) {
10 System.out.println(1);
11 int zahl=4;
12 while(zahl>0)
13 zahl--;
14 }
15 public void aMethod(){
16 //doSomething
17 }
18 }
Roster
Shared Project Session
Chat View
Breakpoints
Debug
user_a@abidjan (Driver)
You
Get Breakpoints from host


Java - PB/src/pa1/Ka1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Saros Run Window Help
Java
Reso
Pa
Hie
PB
Ka1.java
1 package pa1;
2
3 import java.util.ArrayList;
4
5 public class Ka1 {
6 @SuppressWarnings("unchecked")
7 ArrayList a = new ArrayList(5);
8
9 public static void main(String[] args) {
10 System.out.println(1);
11 int zahl=4;
12 while(zahl>0)
13 zahl--;
14 }
15 public void aMethod(){
16 //doSomething
17 }
18 }
Roster
Shared Project Session
Chat View
Breakpoints
Debug
Ka1
Ka1 [line: 13] [conditional] - main(String[])
Ka1 [hit count: 2] [entry] - aMethod()
Ka1 [access and modification] - a

Eintritt in Launch-Phase

- Host initiiert Start des lokalen Debuggers und sendet Nachricht darüber an Peers
- Dabei speichert er VM Instanz in Proxy. Aufrufe auf dem Proxy werden an diesen delegiert. Dient dazu erzeugte Objekte für nachfolgende Aufrufe durch Peers zu cachen und kann zum Logging von Aufrufen eingesetzt werden.
- Peer empfängt Startmeldung und führt Start nach Muster Remote Application Launch durch. Erstellt VirtualMachine über Proxy. Delegate dieses Proxys sendet bei Methodenaufrufen Nachrichten an den Host und blockiert bis dieser ihm antwortet. Der Host antwortet stets mit serialisierten Proxy-Instanzen, damit diese auf Peer Seite zu weiteren Remote Aufrufen führen.
- Peer benötigt Launchdatei aus einem früheren lokalen Debuggerstart. Diese wird um suffix _proxy ergänzt und im Remote Launch verwendet.
- Prinzip von RPC muss mit XMPP Protokoll umgesetzt werden. Dabei Caching von z.B. erstellten EventRequest Objekten aus der JDI Spezifikation oder für effizientes wiederholtes Abfragen von Variablen

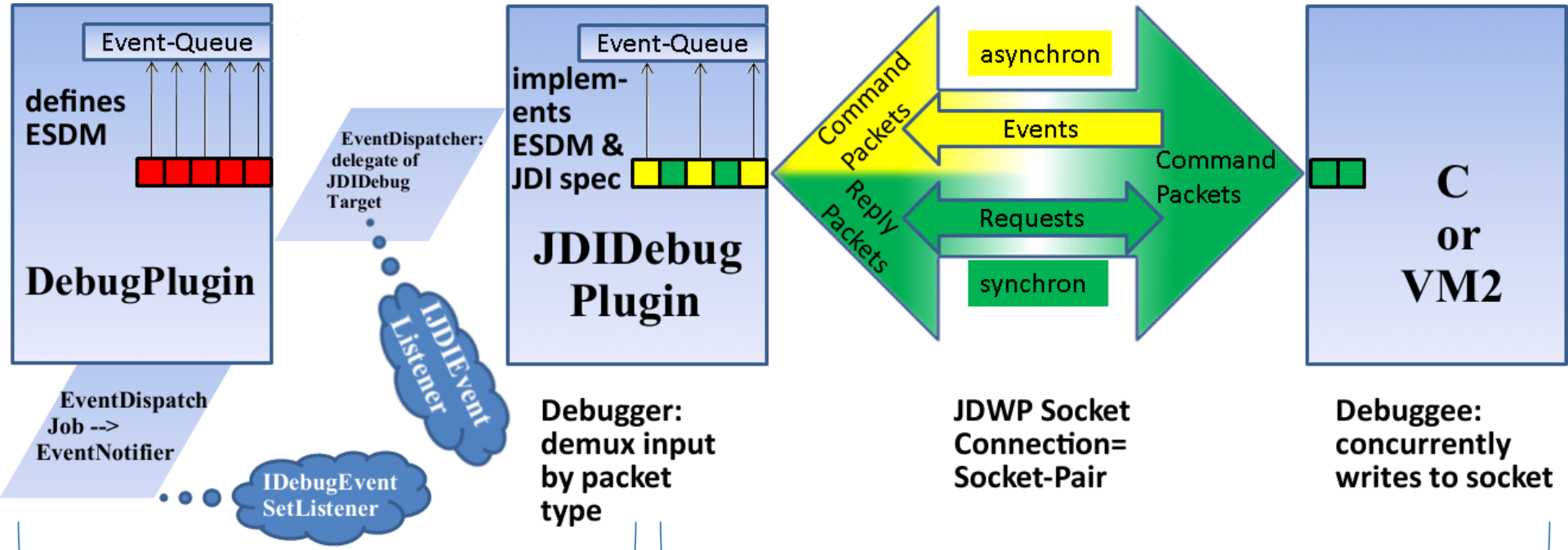

Java - PA/src/pa1/Ka1.java - Eclipse Platform
File Edit So
Debug Configurations
Create, manage, and run configurations
Debug a Java application
PA
type filter text
Java Applet
Java Application
Ka1
Remote Java Applicatic
Filter matched 4 of 4 items
Name: Ka1
Saros Debug Tab
Main
Arguments
JRE
Classpath
Source
Check this box if you want a distributed debug session.
Apply
Revert
Debug
Close
Reso
Start
Startseit...
TeXnicC...
main-tex...
Java - or...
graphics
2 Jav...
hostNoB...
22:34


Java - PB/src/pa1/Ka1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Saros Run Window Help
Java
Reso
Pa
Hie
PB
Ka1.java
1 package pa1;
2
3 import java.util.ArrayList;
4
5 public class Ka1 {
6 @SuppressWarnings("unchecked")
7 ArrayList a = new ArrayList(5);
8
9 public static void main(String[] args) {
10 System.out.println(1);
11 int zahl=4;
12 while(zahl>0)
13 zahl--;
14 }
15 public void aMethod(){
16 //doSomething
17 }
18 }
Roster
Shared Project Session
Chat View
Breakpoints
Debug
Connected as user_b@abidjan/Saros
Buddies
user_a@abidjan Connected using: SOCKS5 (direct)
user_c@abidjan
Launching Ka1_proxy: (57%)
Start
Startseit...
TeXnicC...
main-tex...
Java - S...
graphics
2 Jav...
hostLau...
22:41

Idee des Ansatzes

- In JDI wird ein EventRequest Objekt indirekt vom VirtualMachine Objekt erzeugt.
- Dieses Objekt wird vom EventDispatcher zur Zuordnung von Listenern zu zukünftigen JDIEvents benutzt.
- Wenn ein Peer nun ein Proxy EventRequest Objekt vom Host empfängt, dann kann dessen EventDispatcher dieses bei seiner Zuordnung von Listenern zu JDI Events benutzen.
- Der Host muß nur noch die Eventset Objekte (um Proxy Instanzen ersetzt) an die EventDispatcher der Peers weitersenden
- Die lokalen EventDispatcher benachrichtigen daraufhin die jeweiligen JDIEventListener und diese erstellen oder verändern Instanzen der implementierten ESDM Klassen (EJDM = JDIDebugModel)

Java Debugger in Eclipse



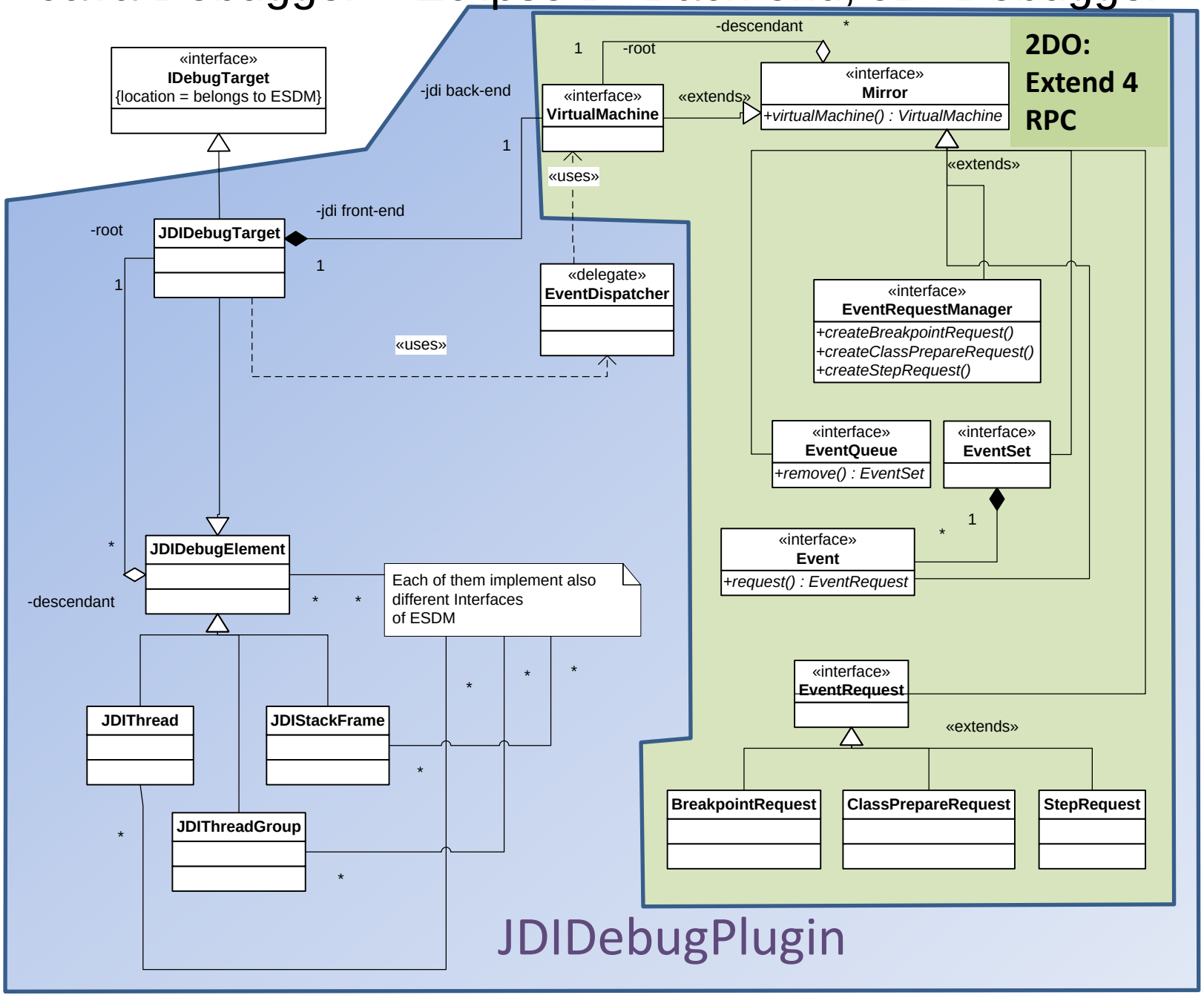
Eclipse Debug Platform

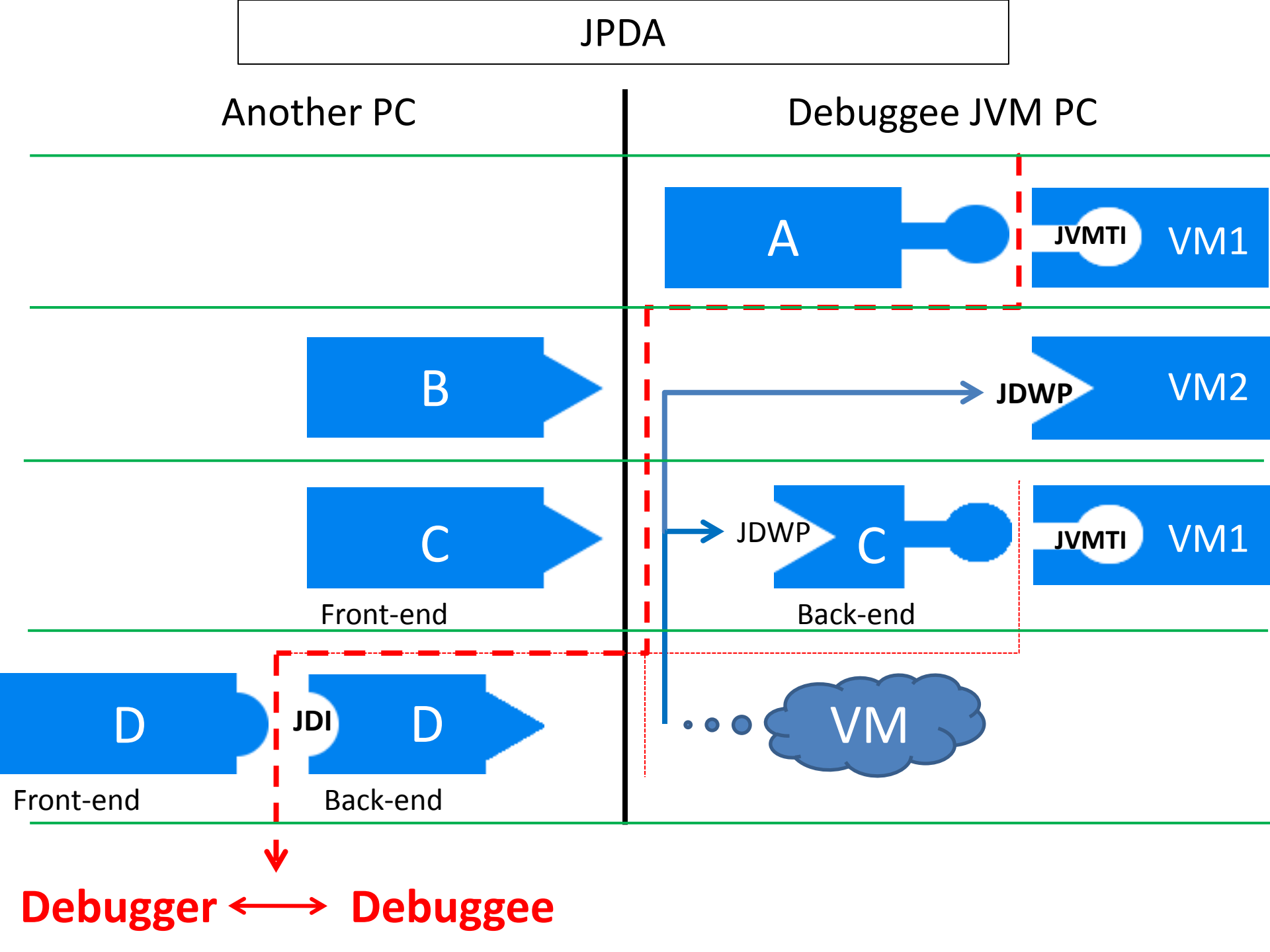
Java Platform Debugger Architecture
JPDA

JDI-Backend RPC fit machen

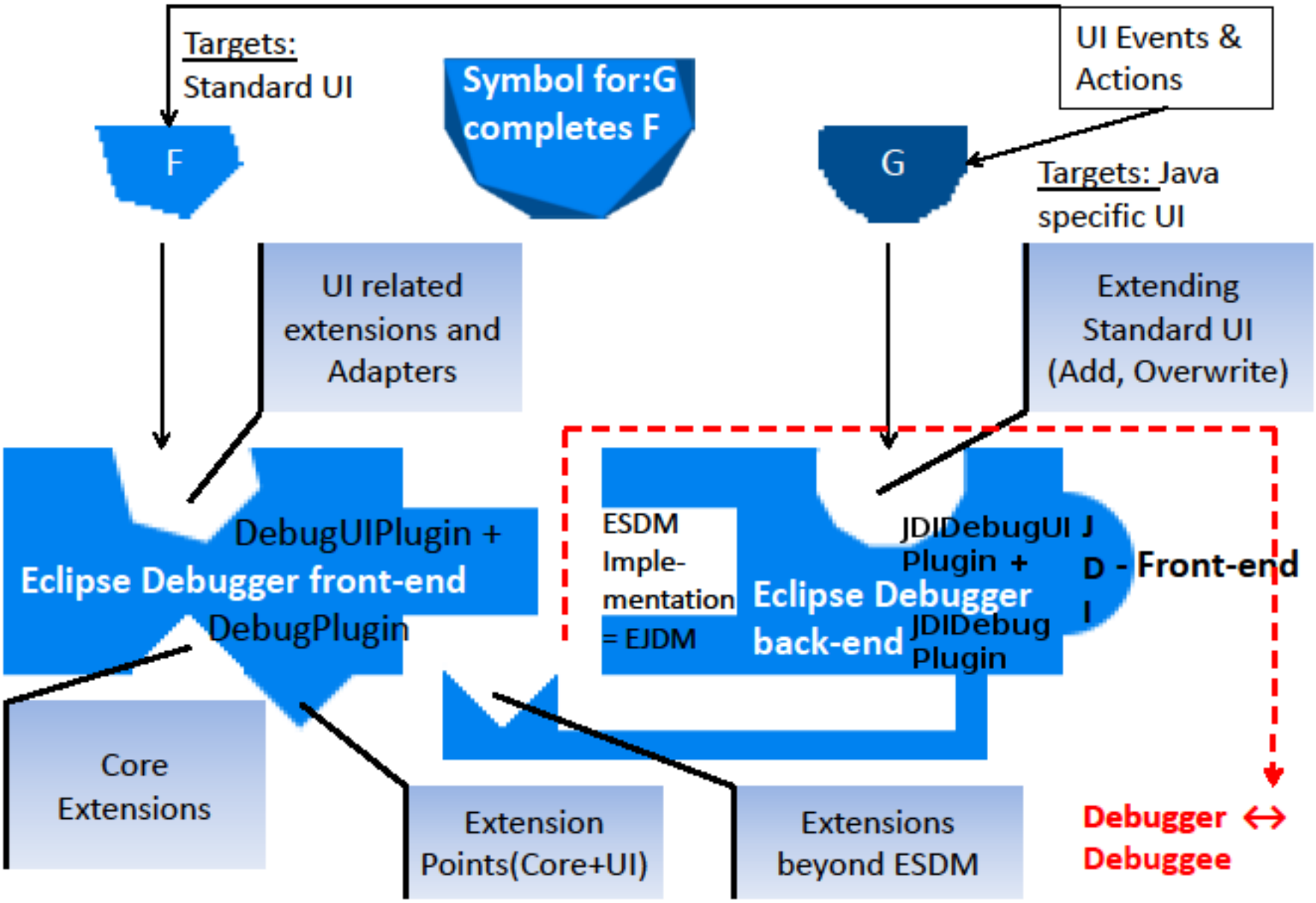
- Javas Dynamische Proxys Framework einsetzen, um darauf aufbauend RPC Funktion in JDI zu integrieren. Protokoll: XMPP
- Alle Instanzen von JDI Klassen müssen bei initialer Referenzierung beim Host in Proxy-Instanzen eingeschlossen werden. Dies gilt auch für Kindelemente.
- Proxy Instanzen werden Peers serialisiert zugesandt. Methodenaufrufe auf diesen führen zu Nachrichten an Host und blockieren bis Antwort zurückkommt.

Java Debugger = Eclipse D.-Back-end, JDI Debugger

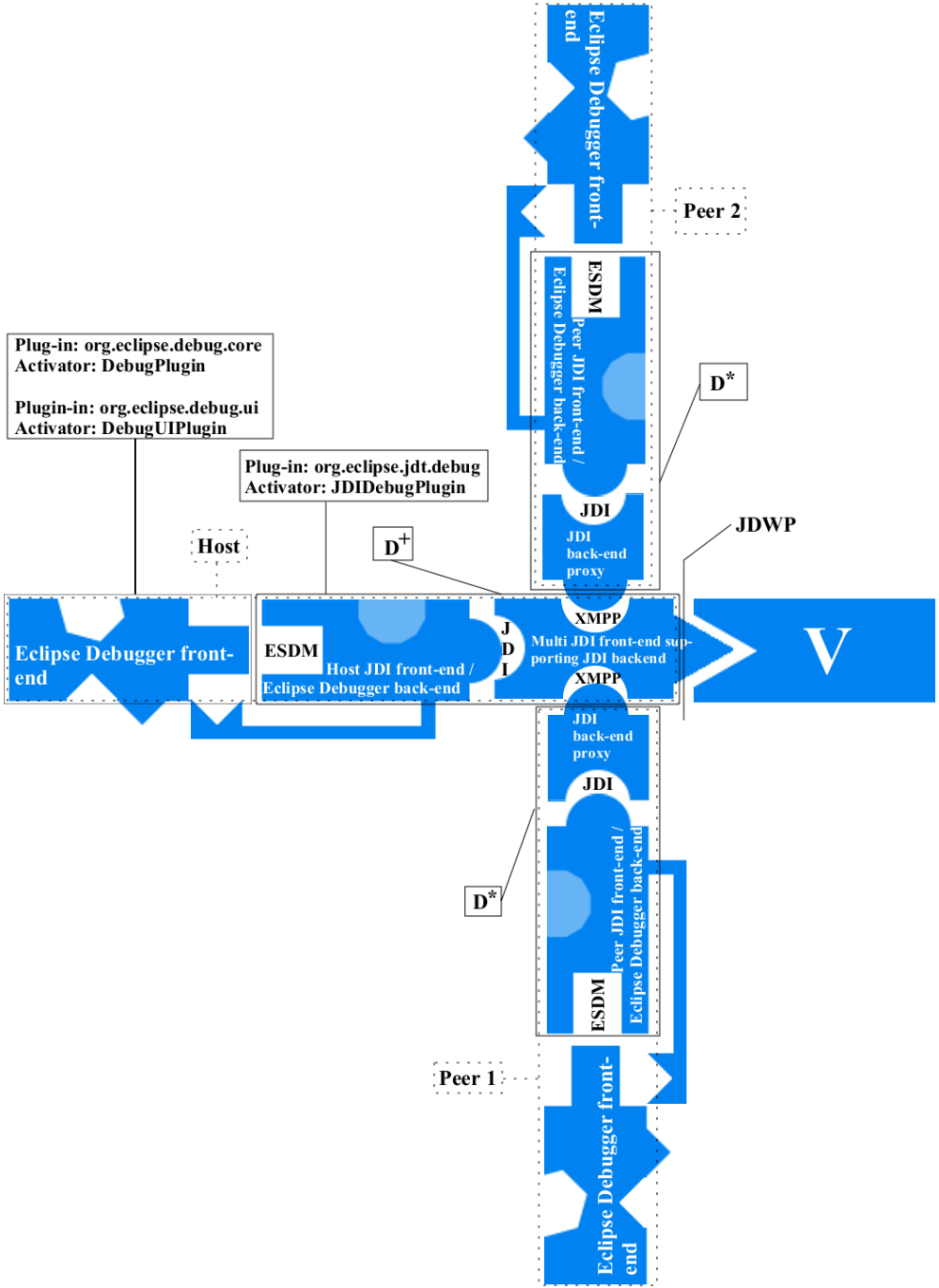



JPDA
Another PC
Debuggee JVM PC
A
JVMTI
VM1
B
JDWP
VM2
C
JDWP
C
JVMTI
VM1
Front-end
Back-end
D
JDI
D
VM
Front-end
Back-end
Debugger
Debuggee

JPDA Integration in Eclipse



Multi-User Java Debugger



Ergebnisse

- Serialisierung von Breakpoints generisch implementiert. Daher auch für beliebige Debugger in Eclipse einsetzbar.
- Pre-Launch nahezu vollständig implementiert. Es müßten noch Modify Events außer ToggleEnable übertragen werden.
- Launch Phase ist schematisch implementiert. Startvorgang kann bereits repliziert werden. Mithilfe der dynamischen Proxys in Java prinzipiell lösbar. Dabei erleichtert Loggingmöglichkeit den erforderlichen Analyseaufwand.